

操作系统实践指导

Guide to Labworks by WQY

吴清晏 (61522314)

东南大学吴健雄学院

前言

感谢您使用《Guide to Labworks by WQY》!

本文由吴健雄学院 22 级吴清晏编写, 希望能帮助同学们顺利完成操作系统实验的准备工作, 加深对操作系统的理解。

首先, 为使用 Windows 系统的同学提供了 VMware 虚拟机和 WSL (Windows Subsystem for Linux) 的安装教程, 还包括实验文件与 Xv6 的下载以及运行环境配置。

然后, 以 Reverse 实验为例, 说明了 C 语言实验的一般流程, 包括需求分析, 功能实现和测试等。

最后, 以 Syscall 实验为例, 说明了 Xv6 实验的一般流程, 包括 Xv6 系统的文件介绍等。

值得注意的是, Mac 系统自带 Unix 内核, 不需要额外配置, 但 Mac 系统的终端和 Linux 系统的终端有一些不同, 需要注意一些命令的区别。据同学所说, Mac 电脑上 Xv6 的运行可能存在一些问题, 主要是因为安装 qemu 存在一定的困难。即使如此, 考虑到 Mac 电脑的性能, 我依旧不推荐使用虚拟机。

目录

1	课前准备教程	1
1.1	虚拟机配置	1
1.2	GNU 编译环境安装	2
1.3	文本编辑器或 IDE 安装	2
1.4	Git 配置	4
1.5	其他配置	4
2	实验文件下载	6
2.1	Xv6 实验	6
3	实验指南	7
3.1	Reverse 实验	7
3.2	Syscall 实验	10

1 课前准备教程

在开始前，请明确你是仅仅想完成《操作系统》的课程实验，还是想获得 Linux 的完整体验。考虑到大部分同学已经习惯了 Windows 系统的操作，我不推荐安装双系统，可通过 Windows 上的 VMware 体验带有图形桌面的 Linux 系统。VMware 的性能有限，且在网络，硬件驱动等方面存在一定的配置难度，目前性能更高的解决方案是使用 WSL (Windows Subsystem for Linux)，但其不包含图形化桌面，大部分操作需要通过命令行进行，如果没有足够的信心，建议还是使用 VMware。无论如何你都会需要的部分有：GNU 编译环境安装，实验文件下载，Xv6 配置。

1.1 虚拟机配置

在配置虚拟机前，需要先选择 Linux 发行版。推荐使用 Ubuntu，它是最流行的 Linux 发行版之一，有着丰富的社区支持和软件资源。如果你对 Linux 有一定的了解，可以选择其他发行版，如 Debian, Fedora 等。对于 WSL，只推荐其明确支持的发行版，其他发行版可能会出现一些问题，如 CentOS 的较早版本会缺少一些关键的库。

1.1.1 VMware

如果你需要运行带有图形界面的软件，那么 VMware 可以提供接近双系统的体验。网上有大量 VMware 的安装教程。此外，还需要下载 .iso 格式的 Linux 安装镜像。推荐通过东南大学镜像站下载，建议选择 *Ubuntu22.04* 版本。使用 VMware 的同学可跳过 Git 配置部分和其他配置部分。

1.1.2 WSL + Ubuntu

目前仅 Win10/11 可使用 WSL，其中 Win11 的配置较简单。以下是一般情况下的配置步骤 [2]：

1. “Win + r” 打开“运行”，输入“control”打开“控制面板”，搜索或在“程序和功能”中选择“启动或关闭 Windows 功能”，勾选“适用于 Linux 的 Windows 子系统”和“虚拟机平台”。
2. “运行”中输入“cmd”打开终端，在终端中输入“wsl --update”。Win11 中会自动安装 Ubuntu，Win10 中需要输入“wsl.exe --list --online”列出所有可用发行版，推荐选择 Ubuntu。安装成功后，Win10 与 Win11 操作没有区别。
3. 安装过程中，根据提示输入用户名和密码。用户名中不能出现英文大写，**密码默认隐藏！**
4. 安装完成后，通过“sudo passwd”设置管理员密码，推荐采用与当前用户一样的密码，防止遗忘。为了防止误操作，平时尽量使用普通用户，需要管理员权限时使用“sudo <command>”。
5. WSL 默认安装到 C 盘，可通过在 Windows 终端中输入以下代码迁移到指定位置：

```
wsl --shutdown # 关闭所有虚拟机
wsl --export Ubuntu <path> # 导出虚拟机到指定位置
wsl --unregister Ubuntu # 卸载原虚拟机
wsl --import Ubuntu <path> <new-path> # 从刚刚位置导入虚拟机
```

以上步骤同样可用于保存虚拟机备份和在不同设备间迁移虚拟机。

Tips:

- 如果近期重新安装过“Win11 家庭版”系统，有可能报错注册表缺失，最简单的方法是升级到 Win11 专业版，安装完成后在“启动或关闭 Windows 功能”页面勾选“Hyper-V”，关闭并重新打开两个功能。
- 部分情况下 Ubuntu 默认登录为 root 用户。可通过以下操作创建有“sudo”权限的普通用户，并设为默认用户。该设置在下次启动 WSL 时生效。
 1. 通过“passwd”命令设置 root 密码。
 2. 通过“adduser <username>”添加用户。
 3. 通过“adduser <username> sudo”给新用户添加管理员权限。
 4. 在 Windows 终端输入“ubuntu config --default-user <username>”

1.2 GNU 编译环境安装

1. 输入“sudo apt-get update”更新软件源列表
2. 输入“sudo apt-get install build-essential gdb”安装 GNU 编译环境和调试工具

Tips:

- build-essential 包括了 gcc, g++ 和 make, 其中 gcc 和 g++ 分别为 c 和 c++ 的编译器, make 可以编译带有 makefile 文件的开源软件代码, 后续将安装的 qemu 就通过 make 编译并启动。
- GDB 的全称是 GNU Debugger, VS code 等 IDE 提供的断点调试等功能就是基于 GDB 的。

1.3 文本编辑器或 IDE 安装

文本编辑器仅支持最基本的功能，如代码高亮和自动格式化等。IDE 可以运行和调试代码，但过于复杂。VS code 介于两者之间，其功能比文本编辑器强大，但学习成本比 IDE 低。更重要的是，WSL 原生支持 VS code，更符合 Windows 使用习惯。

1.3.1 文本编辑器

Ubuntu 系统已默认安装了 vim 和 nano，在终端中输入“vim”或“nano”即可打开。vim 是一个上限很高的文本编辑器，但仅支持键盘控制，习惯鼠标操作的 Windows 用户可能很难适应。nano 是一个更简单的文本编辑器，所有需要的快捷键会显示在页面底部，适合初学者使用。

这两款文本编辑器都是面向命令行的，面向图形界面的文本编辑器往往与图形界面共同安装，但 WSL 也支持 Gnome 文本编辑器等图形界面内置工具。

Gnome 文本编辑器的安装方式是“sudo apt install gnome-text-editor -y”，安装成功后可通过“gnome-text-editor <file-name-or-path>”打开指定文件。该文本编辑器类似 Windows 上的记事本，支持鼠标操作，但功能较为简单。

1.3.2 Visual Studio Code

VS Code 是一款由微软开发的轻量级代码编辑器，支持多种编程语言，拥有丰富的插件和主题。安装步骤如下：

1. 先在 Windows 中安装 VS code，推荐选择“System Installer”版本。安装时建议勾选“添加到右键菜单”。安装完成后，打开 VS code，点击左侧扩展按钮，搜索“Chinese”，安装中文语言插件。网络上有大量 VS Code 的配置教程，这里不做详细介绍。
2. 在 wsl 中输入“code .”，注意中间有空格。这行命令在当前目录下打开 VS Code，首次运行会自动完成 VS Code 的初始化。
3. 在 VS Code 窗口的顶部菜单选择“文件”→“打开文件夹”，输入“~”，“~”代表个人目录。
4. 通过 VS Code 左侧工具栏的资源管理器创建文件“helloworld.c”，输入以下代码：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello World\n");
    return 0;
}
```

保存后，在左侧工具栏选择“运行和调试”，选择 (GDB/LLDB) C/C++，点击绿色的三角形运行代码。期间，需要根据提示安装 C/C++ 扩展。运行时，会自动打开底部集成终端，终端已经打开了两个窗口：

(a) C/C++: gcc 生成活动文件

* 正在执行任务: C/C++: gcc 生成活动文件

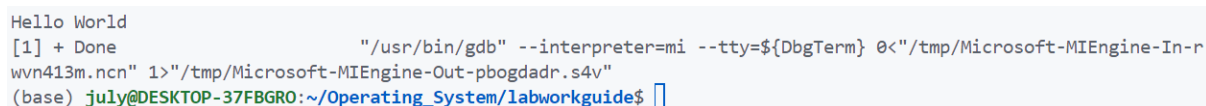
正在启动生成 ...

/usr/bin/gcc -g /home/july/helloworld.c -o /home/july/helloworld

生成已成功完成。

* 终端将被任务重用，按任意键关闭。

(b) cppdbg: helloworld



```
Hello World
[1] + Done                  "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-r
wvn413m.ncn" 1>"/tmp/Microsoft-MIEngine-Out-pbogdadr.s4v"
(base) july@DESKTOP-37FBGRO:~/Operating_System/labworkguide$
```

图 1: cppdbg: helloworld

如果输出与上图相同，说明 GCC 编译器和 GDB 调试器已经安装成功，VS Code 的 C/C++ 扩展也已经安装成功。

1.4 Git 配置

Git 是一个分布式版本控制系统，我们的 Xv6 下载就是通过 Git 进行的。Git 的安装步骤如下：

1. 在 Windows 上，下载 *Git*，安装时建议全部保持默认选项。
2. 如果后续希望在 GitHub 上托管代码，还需要配置用户名和邮箱。在终端中输入以下命令：

```
git config --global user.name "Your Name"
git config --global user.email "youremail@domain.com"
```

3. 在 wsl 中输入 “sudo apt-get install git” 安装 git。

1.5 其他配置

这一部分是非必须的，但是有可能使后续操作更简单，看大家的需求。

1.5.1 将 VScode (WSL) 添加到右键菜单

在文件资源管理器中可右键选择通过 Windows 上的 VScode 打开当前文件夹。但是，对于 WSL 虚拟机中的文件，我们希望使用 WSL 上的 VScode 打开。此外，对于 Windows 11，许多类似选项被放在了“更多选项”中，个人更习惯 Windows7 风格的右键菜单。可通过修改注册表自定义右键菜单，修改前建议备份注册表，具体步骤见教程。

1.5.2 在 WSL 中安装 Chrome 浏览器

在实验文件中，有一些说明文档是 html 格式的，可以通过 VS code 查看，也可以选择文件资源管理器中打开文件路径后，通过 Windows 上的浏览器打开。但是，如果一定想使用 WSL 中的浏览器，可根据官方提供的教程操作。安装完成后，先不要着急打开浏览器，需要配置中文字体。如果现在直接打开，会出现大量警告信息，但配置完中文和输入法后大部分警告消失。剩余警告与 Linux 系统的省电模式有关，运行 `sudo apt-get install upower` 后警告完全消失 [1]。

1.5.3 配置中文与中文输入法

如果你不习惯阅读英文报错信息，可以将 Ubuntu 的命令行语言设为英文 [3]。这不会影响输入命令的语法，只会影响一些报错和帮助信息的部分语句。你还可以安装中文输入法，并在刚刚安装的浏览器中使用。如果你没有安装任何 WSL 上的 GUI 应用，那么你不需要安装任何中文输入法。

1. 输入 “sudo apt install language-pack-zh-hans” 安装中文语言包
2. 输入 “sudo dpkg-reconfigure locales”，在弹出窗口中选择 en_US.UTF-8 和 zh_CN.UTF-8。空格键选择，回车键确认，ESC 键退出。下一步的默认语言就是 Ubuntu 的命令行语言，推荐选择中文 (zh_CN.UTF-8)
3. 输入 “sudo apt-get install fontconfig” 安装字体配置工具

4. 将 Windows 的字体文件链接到 Ubuntu，具体方法是：

- (a) 输入 “`sudo vim /etc/fonts/local.conf`” 创建配置文件，按 “i” 进入编辑模式，输入以下内容：

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<fontconfig>
  <dir>/mnt/c/Windows/Fonts</dir>
  <dir>/usr/share/fonts</dir>
  <dir>/usr/local/share/fonts</dir>
  <dir prefix="xdg">fonts</dir>
</fontconfig>
```

- (b) 按 “ESC” 退出编辑模式，输入 “:wq” 保存并退出

5. 输入 “`fc-cache -f -v`” 刷新字体缓存，并通过在 Windows 终端中输入 `wsl --shutdown` 和 `wsl` 来重启 Ubuntu。注意，如果电脑中包含不止一个 wsl 环境，可以通过 `wsl --set-default <distro>` 设置默认环境，设置后可通过 “wsl” 命令打开默认环境。

6. 输入 “`sudo apt install fcitx dbus-x11 im-config fcitx-sunpinyin adwaita-icon-theme-full`” 安装小企鹅输入法。

7. 输入 “`sudo vim ~/.profile`”，在文件末尾添加以下内容，退出方式与刚刚相同。

```
export GTK_IM_MODULE=fcitx
export QT_IM_MODULE=fcitx
export XMODIFIERS=@im=fcitx
export DefaultIMModule=fcitx
fcitx-autostart &>/dev/null
```

8. 输入 “`source ~/.profile`” 应用配置。

9. 输入 “`fcitx-config-gtk3`”，现在应该能正常看到下图所示信息，说明配置成功。

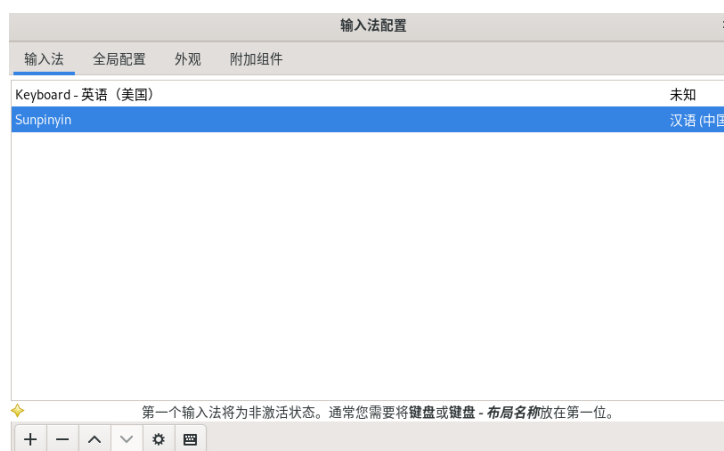


图 2: fcitx 配置信息

2 实验文件下载

在下载实验文件前，请确保网络畅通，且可正常访问 Github。如果无法稳定访问 Github，可尝试“<https://gitclone.com/>”等镜像站点。

2.1 Xv6 实验

1. 从董恺老师提供的链接下载实验文件夹，解压缩到 WSL 中的个人目录下 (如 `~/projects`)。
2. 按照实验指导书的要求，完成普通实验，如 Reverse。
3. 对于 Kernel-Hacking 实验，在 VS code 集成终端中输入“`cd ~/projects/Xv6-Syscall`”。
4. 输入“`git clone https://github.com/mit-pdos/xv6-public.git`”获取实验文件，网络顺畅的话，当前目录下会增加一个名为 `xv6-public` 的文件夹，将其改名为“`src`”。
5. 输入“`objdump -i`”测试编译工具。若按照操作流程完成了 GNU 编译环境安装，应获得如下输出（语言可能存在差异，但请确保输出包含“`elf32-i386`”）。

```
● (base) july@DESKTOP-37FBGR0:~/Operating_System/labworkguide/latex版说明$ objdump -i
BFD 头文件版本 (GNU Binutils for Ubuntu) 2.38
elf64-x86-64
  (header 小端序, data 小端序)
  i386
elf32-i386
  (header 小端序, data 小端序)
  i386
elf32-iamcu
  (header 小端序, data 小端序)
```

图 3: 测试编译工具输出

6. 输入“`sudo apt-get install qemu-system`”安装 qemu 虚拟机
7. 输入“`cd src`”进入文件夹，再输入“`make`”编译。编译完成后，输入“`make qemu`”运行 Xv6 系统。注意，以上过程均应在普通用户状态下执行，如果在 root 用户的个人目录下，会出现意料之外的权限问题。

3 实验指南

本部分旨在对《*Guide to Xv6 Labworks*》中提到的两个实验进行说明，以帮助同学们了解实验的一般流程。希望大家先独立思考，如果遇到无法解决的问题，再尝试通过我的教程解决。

3.1 Reverse 实验

在 VS code 中打开 “~” 目录，在 “./projects/Reverse” 下新建 “reverse.c” 文件。

3.1.1 需求分析

1. 支持 3 种输入形式：

- (a) `./reverse`
- (b) `./reverse input.txt`
- (c) `./reverse input.txt output.txt`

2. 对于输入的数据 (命令行/文件)，**不能**假设句子长度和句子个数。

3. 处理以下 4 种错误：

- (a) 输入参数过多：“usage: reverse <input> <output>”
- (b) 文件无法打开：“reverse: cannot open file <filename>” (<filename> 为打不开的文件名)
- (c) 输入相同文件：“reverse: input and output file must differ” (不能仅通过文件名判断)
- (d) 内存分配失败：“malloc failed”

用 “`fprintf(stderr, <error message>\n);`” 输出错误并 “`exit(1);`” 返回状态码 1。

Tips:

- 其中，`stderr` 是一种特殊的输出流，与之类似的输出流是 `stdout`，`stdout` 类似 `c++` 中 `cout`。
- 返回的状态码正常为 0，调用 `exit` 函数会立即终止并返回指定状态码。

3.1.2 功能实现

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
```

1. 处理错误 “输入参数过多”

```
// reverse 参数过多时打印 usage: reverse <input> <output> 并返回 1
if (argc > 3) {
    fprintf(stderr, "usage: reverse <input> <output>\n");
    exit(1);
}
```

2. 处理错误 “文件无法打开”

```
// 输入流和输出流指针
FILE *input = stdin, *output = stdout;
// 尝试打开输入文件
if (argc >= 2) {
    input = fopen(argv[1], "r");
    if (input == NULL) {
        fprintf(stderr, "reverse: cannot open file '%s'\n", argv[1]);
        exit(1);
    }
}
// 尝试打开输出文件
if (argc == 3) {
    output = fopen(argv[2], "w");
    if (output == NULL) {
        fprintf(stderr, "reverse: cannot open file '%s'\n", argv[2]);
        exit(1);
    }
}
```

3. 通过头文件 “<sys/stat.h>” 提供的 stat 函数处理错误 “输入相同文件”

```
struct stat stat1, stat2;
stat(argv[1], &stat1);
stat(argv[2], &stat2);
if (stat1.st_ino == stat2.st_ino) {
    fprintf(stderr, "reverse: input and output file must differ\n");
    exit(1);
}
```

Tips:

- stat 函数可获取文件的状态信息，其中 st_ino 是文件的 inode 号，用于唯一标识文件。这样可以处理不同文件名对应相同文件的情况，如软链接。
- 在 C 语言中，函数传值常常通过引用参数实现，这与 Python 等语言使用多个返回值的习惯不同。这种方式的优点是可以减少内存开销，缺点是可能会使代码更难理解。

4. 处理错误 “内存分配失败”

```
// 记录行数和初始容量
int capacity = 10;
char **lines = malloc(capacity * sizeof(char *));
if (lines == NULL) {
    fprintf(stderr, "malloc failed\n");
    exit(1);
}
```

5. 实现缓冲区满自动扩容

```
int num_lines = 0,
size_t len = 0;
while (1) {
    if (num_lines == capacity) {
        capacity *= 2;
        lines = realloc(lines, capacity * sizeof(char *));
        if (lines == NULL) {
            fprintf(stderr, "malloc failed\n");
            exit(1);
        }
    }
    if (getline(&lines[num_lines], &len, input) == -1)
        break;
    num_lines++;
}
```

Tips:

- “getline” 函数在 “len” 设置为 0 时，会自动扩充输入缓冲区并更新 “len” 参数。
- 如果想测试零参数下的效果，可通过 “Ctrl+D” 终止输入流，此时 “getline” 函数会返回-1。

6. 将所有句子逆序放入输出流，释放内存并关闭文件

```
for (int i = num_lines - 1; i >= 0; i--) {
    fprintf(output, "%s", lines[i]);
    free(lines[i]);
}

free(lines);

if (input != stdin)
    fclose(input);
if (output != stdout)
    fclose(output);
```

Tips:

- 用 malloc 分配的内存一定要主动调用 free 函数进行内存释放，以防止内存泄漏。在程序终止时，操作系统会自动回收所有已分配资源，但主动回收有助于养成更好的代码习惯。
- 文件操作完成后一定要调用 fclose 函数关闭文件，因为每个打开的文件会消耗一个文件描述符，文件描述符达到上限将无法打开新文件。
- 当你发现电脑上有程序在运行过程中内存占用越来越大，说明发生了内存泄漏，可尝试终止程序并重新打开，利用操作系统完成资源回收。

3.1.3 编译文件并测试功能

1. 选中文件，右键“在集成终端中打开”
2. 输入“gcc -o reverse reverse.c -Wall”进行编译
3. 输入“sudo chmod 777 test-reverse.sh”对当前测试脚本的权限进行修改
4. 输入“sudo chmod 777 ../tester/*”将其他测试脚本的权限设为最高

Tips:

- “-Wall”参数用于在编译时显示所有的警告信息，包括未使用的变量、未初始化的变量、可能的数组越界等潜在问题，有助于提高代码质量。
- 可以用“chmod +x”替代“chmod 777”，前者只添加了执行权限。

5. 输入“./test-reverse.sh”进行测试。如果一切顺利的话，你会看到以下结果：

```

● (base) july@DESKTOP-37FBGR0:~/Operating_System/labworkguide/labwork/Reverse$ ./test-reverse.sh
test 1: passed
test 2: passed
test 3: passed
test 4: passed
test 5: passed
test 6: passed
test 7: passed
○ (base) july@DESKTOP-37FBGR0:~/Operating_System/labworkguide/labwork/Reverse$

```

图 4: 测试结果

3.2 Syscall 实验

打开下载实验文件时创建的“src”文件夹，可看到“makefile”和大量“.c”，“.h”，“.S”文件，在“make”编译后额外出现了“.d”和“.o”文件。“S”或“.s”是汇编文件的后缀名，其中“.S”后缀的文件支持预处理命令(如“#”开头的大写命令)。

在进行系统调用的过程中，我们主要关心的文件有 [4]：

文件名	功能简介
usys.S	提供用户态与内核态转换的接口
syscall.h	定义系统调用号
syscall.c	转发用户发起的系统调用到内核
sysfile.c	实现系统调用
sysproc.c	另一种实现系统调用的选项
user.h	定义系统调用的参数传递方式

接下来，我们将逐个分析这些文件，先观察系统原有的系统调用是如何实现的，再模仿原有的系统调用，实现“getReadCount”。当调用“getReadCount”时，返回 read 系统调用的总使用次数。

3.2.1 usys.S

这个文件包含以下 3 个部分：

1. 头文件

```
#include "syscall.h" // 包含每个系统调用对应的编号($SYS_ ## name)
#include "traps.h"    // 提供了($T_SYSCALL)的定义，此系统中为64。
```

拓展信息：

- \$T_SYSCALL

在 x86 架构的计算机中，系统调用会触发一个陷阱 (trap)。陷阱是一种特殊的中断，它会将 CPU 从用户态切换到内核态，并跳转到对应处理函数，函数地址由中断描述符表 (IDT) 提供，而 \$T_SYSCALL 就是系统中断在 IDT 中的索引。

- IDT

IDT 在 “trap.c” 中定义，是一个长度 256 的 gatedesc 数组。“gatedesc” 在 “mmu.h” 中定义，这是一个结构体，包括 “off_15_0” 和 “off_31_16”，他们合起来表示中断处理程序在其所在段的偏移地址，以及 “cs” (处理函数所在段) 和 “args” (参数数量) 等。

2. 宏定义接口模板

“`.globl name`” 使系统调用的名称成为全局变量，用户程序可以直接通过该名称进行系统调用。当该名称被使用时，自动把 \$SYS_ ## name 放入 “%eax”，并发起 “int” 中断 (“interrupt”)。

```
#define SYSCALL(name) \
    .globl name; \
    name: \
        movl $SYS_ ## name, %eax; \
        int $T_SYSCALL; \
        ret
```

拓展信息：

- %eax

“eax” (Extended Accumulator Register) 是 16 位 “ax” 的 32 位扩展。除 “%eax” 寄存器外，汇编中还有 “%ebx”，“%ecx” 和 “%edx”，但不存在 “%eex” 或 “%efx” 等。实际上，“b” 指 “base” (基底)，“c” 指 “counter”，“d” 指 “data”。

在 x86 架构的 Linux 中，“%eax” 负责传递系统调用的编号和执行完毕时的返回。

- int

大家可能会将其与 c/c++ 中的 int 发生混淆。在汇编语言中，“int n” 中 “int” 为 “interrupt” 指令的缩写，“n” 为中断类型码。中断指令的执行往往包含以下步骤：

- 指令指针 (IP) 和标志寄存器 (FLAGS) 入栈，其中 FLAGS 包含 IF 和 TF
- 查找 IDT 中对应的陷阱门，获取中断处理函数的段选择子和偏移地址
- 将段选择子加载到代码段寄存器 (CS)，将偏移地址加载到指令指针 (IP)
- 执行完成时，通过 “iret” 指令弹出 IP 和 FLAGS，恢复 CPU 状态。

- “IF” (Interrupt Flag)

当“IF”为 1 时，允许响应可屏蔽中断。此处 `int` 指令会将 IF 设为 0。

- “TF” (Trap Flag)

当“TF”为 1 时，CPU 在执行每条指令后生成一个调试异常，通常用于单步调试。此处 `int` 指令会将 TF 设为 0。

3. 通过宏定义模板声明系统调用

已经完成的系统调用均采用“`SYSCALLname`”宏定义，在此基础上实现 `getreadcount`：

```
SYSCALL(getreadcount)
```

注意句尾无分号。

下一步应该检查“`syscall.h`”和“`traps.c`”文件，并根据 `int` 指令的执行过程，寻找下一步须修改的文件。

3.2.2 syscall.h

该文件用于定义系统调用号。在最后增加 `getreadcount(void)` 对应的调用号，如“22”。

3.2.3 trap.c

该文件定义了 `trap` 函数，其开始部分通过检查中断号类型判断是否为系统调用：

```
//PAGEBREAK: 41
void
trap(struct trapframe *tf)
{
    if(tf->trapno == T_SYSCALL){
        if(myproc()->killed)
            exit();
        myproc()->tf = tf;
        syscall();
        if(myproc()->killed)
            exit();
        return;
    }
}
```

当中断类型为系统调用时，将 `trapframe` 信息传递给 `myproc()` 并调用 `syscall()` 函数，该函数在“`syscall.c`”中定义。这里的 `myproc()` 函数通过 `pushcli` 和 `popcli` 调整中断禁用的深度，实现了某种锁，防止在获取当前进程信息时因为调度而错误地获取其他进程的信息。

3.2.4 syscall.c

该文件提供了 `syscall()` 函数：

```
void syscall(void)
{
    int num;
    struct proc *curproc = myproc();

    num = curproc->tf->eax;
    if (num > 0 && num < NELEM(syscalls) && syscalls[num]) {
        curproc->tf->eax = syscalls[num]();
    }
    else{
        cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
        curproc->tf->eax = -1;
    }
}
```

该函数用于将系统调用转发到内核。在该函数中，通过“`tf->eax`”获取系统调用号，通过“`syscalls[num]`”获取对应的系统调用函数，执行该函数并将返回值存入“`tf->eax`”中。

由此可见，我们需要在 `syscalls[]` 中增加 `getreadcount()` 函数的调用：

```
[SYS_getreadcount] sys_getreadcount
```

这时，出现了“未定义标识符“`sys_getreadcount`””报错，继续向上翻，我们还需要添加一行：

```
extern int sys_getreadcount(void);
```

这一行会告诉编译器，我们已经在其他文件中定义了该函数，编译器会自动去寻找。但编译器并没有找到“`sys_getreadcount`”的函数定义，警告依然存在。

其他系统调用在“`sysfile.c`”或“`sysproc.c`”中定义，“`sysfile.c`”负责与文件相关的系统调用，如打开文件、读写文件、关闭文件等，“`sysproc.c`”负责与进程管理相关的系统调用，如创建进程、结束进程、等待进程等。

“`getreadcount`”被调用后会返回“`read`”的调用次数，个人认为它应该被归类为文件操作，但你也可以选择在“`sysproc.c`”中完成函数的定义，对于操作系统来说这两种方式没有区别。

3.2.5 sysfile.c

这个部分留给大家自己完成，你可以定义一个全局变量，每次调用“`read`”时自增，然后在“`getreadcount`”中返回该变量的值。

3.2.6 user.h

这个文件很多人可能会漏掉，它的作用是向用户提供该函数的接口。在其中增加一行：

```
int getreadcount(void);
```

这样能帮助 c 编译器检查系统调用传递的参数是否正确。

3.2.7 测试功能

在文件夹“Xv6-Syscall”中打开集成终端，并修改测试脚本权限：

```
sudo chmod 777 test-getreadcount.sh
./test-getreadcount.sh
```

正常情况下，因为在‘getreadcount’实现部分没有对多线程进行针对性的处理，test2 可能会无法通过，这是正常的。

Tips:

有使用 VMware 的同学在这一步出现了以下报错，可能是系统内置的包不同导致。

```
./testgetreadcount.sh
doing one-time pre-test (use -s to suppress)
../tester/xv6-edit-makefile.sh: line 6: gawk: command not found
make: *** No rule to make target 'clean'. Stop.
make: Nothing to be done for 'xv6.img'.
make: *** No rule to make target 'fs.img'. Stop.
```

解决方法是手动安装缺少的包：

```
sudo apt-get install gawk
```

References

- [1] GitHub. *WSL 2 - Getting “Failed to connect to the bus: Could not parse server address” error when launching google chrome or any electron app*. 2023. URL: <https://github.com/microsoft/WSL/issues/7915#issuecomment-1163333151> (visited on 08/02/2023).
- [2] Microsoft. *WSL 安装教程*. 2023. URL: <https://learn.microsoft.com/zh-cn/windows/wsl/install> (visited on 08/02/2023).
- [3] MonkeyWie. *WSL2 GUI 应用配置实践 - IDEA 篇*. 2021. URL: <https://monkeywie.cn/2021/09/26/wsl2-gui-idea-config/> (visited on 09/26/2021).
- [4] *xv6 Official Tutorial*. <https://github.com/remzi-arpacidusseau/ostep-projects/tree/master/initial-xv6>.